Problèmes d'optimisation combinatoire

L'**optimisation combinatoire**, (sous-ensemble à nombre de solutions finies de l'**optimisation discrète**), est une branche de l'optimisation en mathématiques appliquées et en informatique, également liée à la recherche opérationnelle, l'algorithmique et la théorie de la complexité.

Définition

Dans sa forme la plus générale, un problème d'**optimisation combinatoire** (sous-ensemble à nombre de solutions finies de l'**optimisation discrète**) consiste à trouver dans un ensemble discret un parmi les *meilleurs* sous-ensembles (ou solutions) réalisables, la notion de *meilleure solution* étant définie par une fonction objectif. Formellement, étant donnés :

- un ensemble discret N fini;
- une fonction d'ensemble $f : 2^N \longrightarrow$ R, dite *fonction objectif* ;
- et un ensemble R de sous-ensembles de N, dont les éléments sont appelés les *solutions réalisables*,

un problème d'optimisation combinatoire consiste à déterminer

$\max \{f(S) : S \in R\}$

L'ensemble des solutions réalisables ne saurait être décrit par une liste exhaustive car la difficulté réside ici précisément dans le fait que le nombre des solutions réalisables rend son énumération impossible.

Exemples

- Pour arriver à l'heure au cinéma, quel est le plus court chemin ? À chaque carrefour, à chaque station de bus ou de métro, plusieurs possibilités s'offrent à vous pour poursuivre votre chemin. Les diverses combinaisons dessinent ainsi plusieurs centaines de chemins possibles entre un même point de départ et une même destination. Trouver, parmi toutes ces possibilités, le trajet le plus rapide qui vous permettra de ne pas rater le début du film, c'est résoudre un problème d'optimisation combinatoire.
Rechercher le chemin le plus court ou le plus rapide entre deux points est un problème assez simple à résoudre. Mais si l'on désire faire plusieurs étapes – profiter du trajet pour chercher ses enfants à l'école, passer à la poste, acheter du pain, etc. –, le problème se complique très vite. Quand le nombre d'étapes croît, le nombre de possibilités augmente au-delà de ce que l'on peut énumérer de tête, et même au-delà de ce que peuvent examiner les ordinateurs les plus puissants : c'est l'« explosion combinatoire ».
- Le problème du voyageur de commerce (TSP) où un voyageur de Commerce doit parcourir les N villes d'un pays en effectuant le trajet le plus court. Une résolution par énumération nécessite de calculer ((N-1)!)/2 trajets entre des villes. En particulier, pour

24 villes, il faudrait en générer (23!)/2 environ $2{,}5\times10^{22}$. Pour fournir un ordre de grandeur comparatif, une année ne compte qu'environ $3{,}16\times10^{13}$ microsecondes.

II Problèmes d'optimisation classiques

Les graphes sont un outil de modélisation puissant et très intuitif. Ils sont naturellement utilisés pour représenter des réseaux de transport, de communication, et plus généralement des flux de matières ou d'informations. Ils sont également utilisés pour modéliser des problèmes dans lesquels des objets sont en relation, les sommets du graphe représentant ces objets et les arêtes (ou arcs si une orientation est considérée) les relations entre ces objets. Le problème d'optimisation considéré est modélisé par les outils de la théorie des graphes ensuite une solution est cherchée qui est généralement un algorithme.

Parmi les problèmes modélisés par la théorie des graphes on cite : le problème d'affectation, de flot maximum ou minimum, de transport, de sac à dos, de voyageur de commerce, de localisation et d'ordonnancement etc. Ce dernier regroupe l'ordonnancement dans la gestion de projet et l'ordonnancement dans la gestion de la production. Nous donnerons plus tard les principales caractéristiques ainsi que leur classification avec plus de détails.

III Complexité

A première vue, comment définir un algorithme efficace ? Pour un problème donné, chercher un algorithme efficace, veut dire trouver un algorithme où le temps nécessaire à son exécution ne soit pas trop important.

Un problème est dit facile si on peut le résoudre facilement, c'est-à-dire s'il ne fait pas trop de temps pour arriver à la solution. Donc, s'il existe un algorithme efficace pour un problème donné, alors ce dernier est dit facile.

Un problème pour lequel on ne connaît pas d'algorithme efficace, est ce qu'il est facile ou difficile ?

De nombreux chercheurs se sont penchés sur ce genre de problèmes et ils ont développé une théorie dite de la complexité. Nous n'allons pas détailler cette théorie, mais nous allons, quand même donner une idée globale du sujet en question.

III.1 Complexité d'un algorithme

La définition d'un modèle de calcul a pour but de fournir un mode d'expression à une méthode spécifique de résolution des problèmes.

Dans un modèle de calcul, si un programme destiné à une résolution d'un problème s'arrête pour toutes les données d'entrée de ce problème, alors on dit qu'il est un algorithme.

La complexité d'un algorithme A est une fonction C (A_N), donnant le nombre d'instructions caractéristiques exécutées par A dans le pire des cas, pour une donnée de taille N.

Certains pourraient demander ce qu'est précisément une opération. En général, on considère comme étant une opération élémentaire une affectation, une addition, un test... Mais cela est discutable puisque selon le langage et le compilateur, une opération sera exécutée plus ou moins vite, une addition s'exécutera plus ou moins vite qu'un test... Le nombre d'opérations varie selon les données qu'on a en entrée.

Prenons l'exemple d'un algorithme de tri d'un tableau à une dimension. Il est bien clair que le temps nécessaire pour exécuter cet algorithme avec 5 éléments n'est pas le même avec 500 éléments.

Un autre facteur déterminant dans le temps d'exécution d'un algorithme est la taille des données en entrée. Un tableau à deux dimensions prend plus de temps qu'un tableau à une seule dimension.

Enfin, un dernier facteur considéré dans la complexité est celui de la situation de la structure de données. Par exemple, l'exécution d'un tableau trié au départ n'est pas la même que celui du même tableau désordonné. Donc, la meilleure solution pour ce cas est de prendre le nombre d'opérations dans le pire des cas.

Le critère de rapidité n'est pas toujours celui qui nous intéresse. On peut aussi vouloir estimer la place utilisée par un algorithme dans la mémoire de l'ordinateur. Dans ce cas, on parle de complexité spatiale alors que jusqu'à présent on considérait la complexité temporelle. La complexité spatiale peut se définir d'une manière semblable à la complexité temporelle.

III.2 Algorithme efficace

La complexité d'un algorithme est notée par *O*. Ainsi, si le nombre d'itérations nécessaires pour obtenir une solution optimale est décrit par $4n^3 - 6n^2 + 5n - 1$, l'algorithme est dit en $O(n^3)$, ce qui caractérise une complexité polynomiale en la taille du problème. Par contre si la complexité est de l'ordre $O(2^n)$ on parle de complexité exponentielle. Un algorithme sera dit efficace si sa complexité est bornée par un polynôme ayant la taille des données comme variable.

Considérons les algorithmes A, B et C. Leurs complexités (C) sont les suivantes :

	Complexité	**Nombre d'opérations pour N = 4**	**Nombre d'opérations pour N = 20**
A	$70 * n$	280	1400
B	$9 * n^2$	144	3600
C	$n!$.	24	$2.4 * 10^{18}$

Tableau II.1 Exemple de 3 algorithmes avec leurs complexités.

On remarque que le plus efficace pour 4 éléments est donc C, mais tout de suite, il ne devient plus utilisable. Par contre, A et B restent applicables.

Maintenant, avec 100 éléments, il faut respectivement 8000 et 100000 opérations. Bien évidemment, l'algorithme A est le plus performant, mais l'algorithme B reste applicable. Le nombre d'opérations de C devient presque inimaginable.

Cet exemple justifie la notion d'efficacité. Si le nombre d'opérations "n'explose pas" avec une augmentation de la taille des données, l'algorithme est considéré efficace.

Cependant, il faut bien comprendre que l'on s'intéresse au comportement général de l'algorithme face à des problèmes de grande taille.

Ainsi, ce n'est pas utile de compter toutes les opérations dans le détail, ni de considérer le langage de programmation. Dans notre exemple, les coefficients 70 et 9 ne sont pas très importants, dès que la taille augmente, on s'aperçoit que c'est le terme en *n* qui prime. Ceci explique la difficulté à déterminer la complexité d'un algorithme. Il faut être très précis dans la démarche mais ne pas se soucier de la valeur exacte en terme de temps d'exécution de chaque opération.

III.3 Problème de reconnaissance

Un problème d'optimisation combinatoire (POC) consiste à chercher le minimum s^* d'une application *f*, le plus souvent à valeurs entières, sur un ensemble fini *S* :

$F(s^*) = min \{f(s)\}\ s \in S$

Un problème d'existence (PE) consiste à chercher dans un ensemble fini *S* s'il existe un élément *s* vérifiant une certaine propriété *P*.

Un problème de reconnaissance ou problème de décision est un problème qui consiste à apporter une réponse "oui" ou "non" à une question.

A chaque problème d'optimisation combinatoire (POC), on peut associer un problème de reconnaissance.

Un problème d'optimisation combinatoire est au moins aussi difficile que le problème de reconnaissance associé. En d'autres termes, cela signifie qu'un problème d'optimisation combinatoire est souvent du même niveau de difficulté que le problème de reconnaissance associé. Cela justifie que la suite de ce chapitre ne concerne que les problèmes de reconnaissance.

III.4 Problèmes faciles et problèmes difficiles

*Les problèmes indécidables sont ceux pour lesquels aucun algorithme, quel qu'il soit, n'a été trouvé pour les résoudre***.**

A l'opposé,*les problèmes décidables sont ceux pour lesquels il existe au moins un algorithme pour les résoudre*.

III.4.1 La classe NP

La classe NP est celle des problèmes d'existence dont une proposition de solution est Oui et qui est vérifiable polynomialement. Parmi les problèmes décidables, les plus simples à résoudre sont regroupés dans la classe NP.

La classe NP est également décomposée en trois catégories qui permettent d'identifier les problèmes les plus simples et les problèmes les plus compliqués de la classe.

III.4.1.1 La classe P

Un problème est dit polynomial s'il existe un algorithme de complexité polynomiale permettant de répondre à la question posée dans ce problème, quelle que soit la donnée de celui-ci. La classe P est l'ensemble de tous les problèmes de reconnaissance polynomiaux.

Pour le reste de la classe NP, on n'est pas sûr qu'il n'existe pas un algorithme polynomial pour résoudre chacun de ses problèmes. Ainsi, on sait que P est inclue dans NP mais on n'a pas pu prouver que P n'est pas NP.

III.4.1.2 La classe NP-Complet

La classe NP-Complet regroupe les problèmes les plus difficiles de la classe NP. Elle contient les problèmes de la classe NP tels que n'importe quel problème de la classe NP leur est polynomialement réductible. Entre eux, les problèmes de la classe NP-Complet sont aussi difficiles.

III.4.1.3 La classe NP-Difficile

La classe NP-Difficile regroupe les problèmes (pas forcément dans la classe NP) tels que n'importe quel problème de la classe NP leur est polynomialement réductible.

On peut résumer ce que nous avons vu dans le tableau récapitulatif suivant :

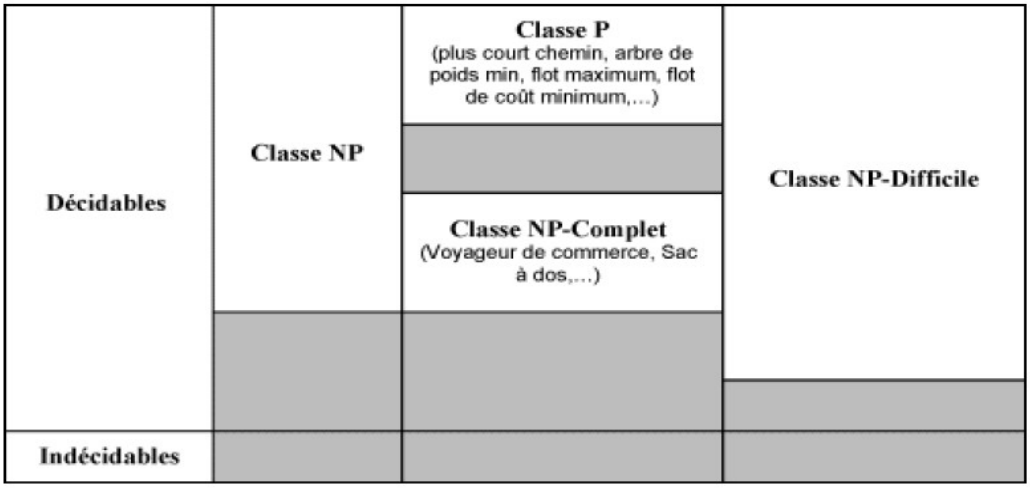


Tableau II.2. Tableau récapitulatif de classification des problèmes combinatoires .

IV. Méthodes de résolution

Historiquement, c'est la recherche opérationnelle qui a commencé à élaborer des modèles d'optimisation programmables. Depuis, d'autres techniques, développées dans le cadre de la résolution de problèmes en Intelligence artificielle, sont venues s'y joindre. Le choix d'une méthode de résolution répond toujours à une certaine problématique que l'on peut tenter de caractériser quatre questions-clés: l'existence d'un modèle d'optimisation, l'exactitude des solutions, le mode de résolution (automatique/interactif) et le coût de la méthode. Ces questions ne sont pas totalement indépendantes.

Dans une démarche basée sur un modèle d'optimisation, l'ensemble des connaissances permettant de fournir une solution idéale est intégré dans un modèle programmable dont l'exécution ne demande aucune intervention - ou presque - d'un décideur. Mais certaines connaissances sont difficiles à modéliser ou à représenter ou à exploiter (surtout dans un milieu aléatoire où les données sont incertaines, imprécises...), il est impératif, dans ce genre de cas, de faire appel à des approches basées sur l'aide à la décision qui sont plus réalistes et plus souples.

Le coût, dans le choix d'une méthode de résolution d'un problème POC, est évidemment très important. Malgré que la puissance des ordinateurs croît de jour en jour et la limite de la taille des problèmes combinatoires recule également mais l'utilisation de certaines méthodes dires exactes restent toujours d'un coût très élevé. Dans le cas d'une résolution interactive, le nombre de décisions prises par le décideur, et/ou la remise en question de ces décisions est également un facteur qui conditionne ce coût sans parler de celui causé par la méthode de résolution elle-même.

Dans ce chapitre, nous allons présenter brièvement l'ensemble des méthodes de résolution utilisées pour résoudre un problème POC qui fait partie d'un domaine plus grand qui est l'optimisation combinatoire en s'inspirant des quatre questions citées un peu plus haut.

IV.1 Méthodes classiques de résolution

Comme nous l’avons signalé précédemment, les problèmes appartenant à la classe P ont des algorithmes efficaces et sont de complexité polynomiale tel que la méthode de

chemin critique, la méthode PERT dans l'ordonnancement de projet... Pour les autres problèmes, il est peu réaliste de trouver des algorithmes de ce type. Alors, on fait appel à une famille de méthodes qui sont **exactes** ou **approchées**.

La notion d'exactitude est relative : une méthode utilisant un critère d'optimisation est exacte si elle garantit l'optimalité des solutions trouvées; sinon elle sera dite approchée, lorsqu'on observe empiriquement qu'elle fournit de "bonnes" solutions. Si l'objectif se réduit à l'admissibilité des solutions, une méthode par caractérisation sera dite exacte si elle fournit une solution satisfaisant toutes les contraintes du problème, y compris celles que constituent les décisions fournies interactivement par un décideur. Une méthode interactive basée sur la propagation de contraintes peut être considérée comme approchée si ses déductions sont incomplètes.

Dans ce qui suit, nous allons passer en revue rapidement les méthodes génériques utilisées d'une façon générale aux problèmes combinatoires et les méthodes dédiées spécialement aux problèmes d'ordonnancement.

IV.2 Méthodes exactes

Les trois familles de méthodes exactes sont : la méthode de séparation et évaluation (Branch and Bound), la programmation dynamique et la résolution à partir d'une modélisation analytique. Ces méthodes se caractérisent par leur coût très élevé en terme de temps de calcul qui est exponentiel. Ces méthodes ne sont appliquées que sur des problèmes de petite taille.

IV.2.1 Méthode de séparation et évaluation

Cette méthode est caractérisée par une énumération implicite et intelligente de l'ensemble des solutions réalisables. La séparation consiste à fractionner l'ensemble des solutions en plusieurs sous-ensembles et qui sont décomposés à leur tour selon une approche itérative. En fin de compte, le processus peut être visualisé en tant qu'un arbre d'énumération et où les nœuds de l'arbre représentent les sous-ensembles et les feuilles correspondent à des solutions réalisables.

Pour pouvoir accéder à la solution plus rapidement en évitant l'exploration inutile de certains nœuds, on applique une procédure d'évaluation qui cherche, à chaque niveau de l'arbre, la valeur de chaque nœud. Cette évaluation permet de décider si ce nœud (sous-ensemble) est à explorer ou non.

Notons que les remontées et les descentes faites dans d'autres nœuds de l'arbre sont effectuées tant que le critère d'arrêt de la procédure d'évaluation n'est pas atteint et que la qualité de ce dernier (la borne du critère) détermine l'efficacité de la méthode en terme de nombre de nœuds explorés.

Ainsi, certaines branches de l'arbre sont évitées de l'exploration, ce qui permet de ne pas énumérer toutes les solutions réalisables.

IV.2.2 La programmation dynamique

Le principe de la méthode, lorsqu'on a un problème de dimension *n*, il est décomposé en *n* sous problèmes de dimension 1. Le processus est constitué de *n* étapes que l'on résout séquentiellement et le passage d'une étape à une autre se fait à partir des lois d'évolution du système et d'une décision.

Le principe d'optimalité tel qu'il est proposé par Bellman (1957) se base sur l'existence d'une équation récursive permettant de décrire la valeur optimale du critère à une étape donnée en fonction de la valeur acquise dans l'étape précédente.

Ainsi, pour appliquer la programmation dynamique (DP) à un problème combinatoire, le calcul du critère pour un sous-ensemble de taille k nécessite la connaissance de ce critère pour chaque sous-ensemble de taille k-1, ce qui porte le nombre de sous-ensembles considérés à 2^n (où n est le nombre d'éléments considérés dans le problème), d'où sa complexité exponentielle. Pourtant, pour les problèmes NP-difficiles au sens faible, il est souvent possible de construire un algorithme de programmation dynamique pseudo-polynomial, pouvant être utilisé pour des problèmes de dimension raisonnable.

IV.3 Modélisation analytique et résolution

Le principe de cette approche est de modéliser le problème sous forme d'un programme linéaire à variables réelles. La littérature scientifique propose plusieurs méthodes de résolution comme par exemple le simplex. Mais, lorsque le problème contient des variables entières, les modèles deviennent beaucoup plus difficiles à résoudre. Il en est de même si le modèle n'est pas linéaire, comme par exemple le cas du modèle quadratique. La programmation par contraintes en fait partie de ce groupe.

IV.3.1 Méthodes approchées

Dans la majorité des cas, où les problèmes sont NP-difficiles de grande taille, les méthodes exactes ne sont pas envisageables à cause de leur temps de calcul exorbitant. Il est préférable de faire appel à des méthodes approximatives qui donnent des solutions acceptables mais non optimales en un temps de calcul raisonnable et polynomial.

Ces méthodes sont généralement itératives où, à chaque itération, une solution intermédiaire est complétée.

IV.3.1.1 Heuristiques

Une heuristique est une règle empirique simple basée sur l'expérience. C'est une technique de calcul approchée qui exploite d'une façon satisfaisant le problème à optimiser et qui fournit une solution admissible, non nécessairement exacte, dans un temps polynomial pour un problème d'optimisation difficile. En d'autres termes, une heuristique est un algorithme qui sacrifie en partie, la qualité de la solution au sens purement mathématique pour accélérer le processus de résolution. Elle est également une **stratégie de bon sens pour se déplacer intelligemment dans l'espace des solutions,** afin d'obtenir une solution approchée, la meilleure possible, dans un délai de temps raisonnable.

Elle est là, à la place d'une méthode exacte qui donne la solution dans un temps de résolution exponentiel. Elle est dédiée, généralement, à un problème bien particulier.

Les heuristiques sont dépendantes du problème à résoudre, principalement dans le choix du voisinage (donc dans le déplacement dans l'espace des solutions).

De nombreuses méthodes heuristiques ont été proposées dans la littérature pour résoudre les problèmes POC. On distingue:

- **FIFO** (First In First Out) : la première tâche qui vient est la première tâche ordonnancée,
- **SPT** (Shortest Processing Time) : la tâche ayant le temps opératoire le plus court est traitée en premier lieu,
- **LPT** (Longest Processing Time) : la tâche ayant le temps opératoire le plus important est ordonnancée en premier lieu,
- **EDD** (Earliest Due Date) : cet algorithme choisit parmi les tâches exécutables celle dont le délai est échu le plus tôt. Si aucune tâche n'est disponible, alors un temps libre est généré,
- **SRPT** (Shortest Remaining Processing Time) : cette règle, servant à lancer la tâche ayant la plus courte durée de travail restant à exécuter, est très utilisée pour minimiser les encours et dans le cas des problèmes d'ordonnancement préemptifs,
- **ST** (Slack Time) : à chaque point de décision, l'opération ayant la plus petite marge temporelle est prioritaire. Faute de disponibilité des ressources de production, cette marge peut devenir négative.

IV.3.1.2 Méta-heuristiques

Le mot méta-heuristique est dérivé de la composition de deux mots grecs: "heuristique" qui vient du verbe "heuriskein" et qui signifie 'trouver' et le mot "meta" qui est un suffixe signifiant 'au-delà', 'dans un niveau supérieur.

Des heuristiques plus poussées ont été mises au point et ont données naissance à une nouvelle famille d'algorithmes : les méta-heuristiques.

Le but d'une méta-heuristique, est de réussir à trouver un optimum global. Pour cela, l'idée est à la fois de parcourir l'espace de recherche, et d'explorer les zones qui paraissent prometteuses ; mais sans être « piégé » par un optimum local.

Les méta-heuristiques sont souvent inspirées de processus naturels.

Il existe un grand nombre de méta-heuristiques différentes, allant de la simple recherche locale à des algorithmes complexes de recherche globale. Ces méthodes utilisent cependant un haut niveau d'abstraction, leur permettant d'être adaptées à une large gamme de problèmes différents.

On peut classer les méta-heuristiques selon leur principe de fonctionnement:

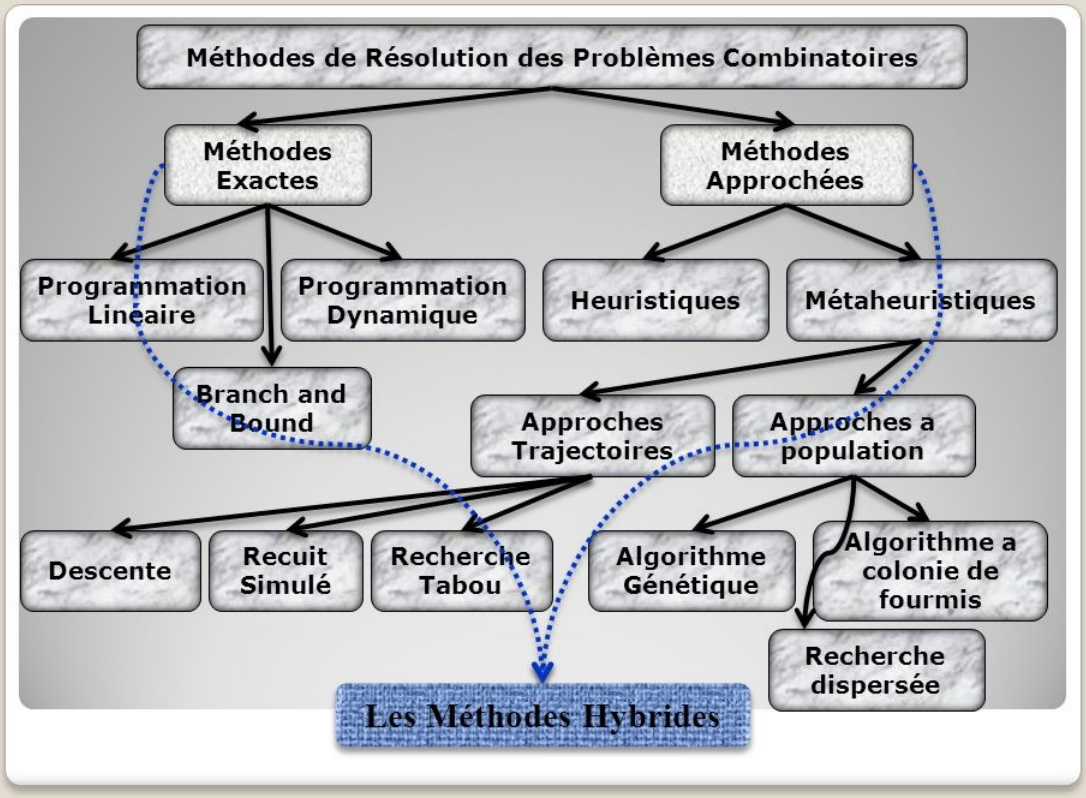
- Par trajectoire (Récuit simulé, recherche tabou,...)
- A base de population (Algorithme génétique, coonie de fourmis,...)

Il existe un très grand nombre d'autres méta-heuristiques, plus ou moins connues :

- l'algorithme du kangourou,
- la méthode de Fletcher et Powell,
- la méthode du bruitage,
- la tunnelisation stochastique,
- l'escalade de collines à recommencements aléatoires,
- la méthode de l'entropie croisée,
- l'algorithme de recherche d'harmonie, etc.

Les méta-heuristiques hybrides sont apparues en même temps que le paradigme lui-même, mais la plupart des chercheurs n'y accordaient que peu d'intérêt. Elles gagnent maintenant en popularité, car les meilleurs résultats trouvés pour plusieurs POC ont étés obtenus avec des algorithmes hybrides. Les méthodes hybrides peuvent être divisées en deux groupes: les méta-heuristiques hybrides qui impliquent une combinaison de plusieurs méta-heuristiques et les méthodes hybrides impliquant une combinaison d'une méthode exacte et d'une méta-heuristique.

En voici un schéma général qui regroupe ces méthodes:


Méthodes de Résolution des Problèmes Combinatoires
Méthodes Exactes
Méthodes Approchées
Programmation Linéaire
Programmation Dynamique
Heuristiques
Métaheuristiques
Branch and Bound
Approches Trajectoires
Approches a population
Descente
Recuit Simulé
Recherche Tabou
Algorithme Génétique
Algorithme a colonie de fourmis
Recherche dispersée
Les Méthodes Hybrides